

ASP.NET Core avec VSCode et Docker

Outils nécessaires

1. Installer le SDK .NET Core : <https://dotnet.microsoft.com/learn/dotnet/hello-world-tutorial>
2. Installer VS Code : <https://code.visualstudio.com/>
3. Installer l'extension VS Code : **ASP.NET core VS Code Extension Pack**
4. Installer le CodeGenerator : `dotnet tool install --global dotnet-aspnet-codegenerator`
5. Installer Docker : <https://www.docker.com/get-started>

Installer et lancer une instance de SQLServer en utilisant Docker

Exécutez la commande suivante (en adaptant les paramètres) :

```
"docker run -d -e SA_PASSWORD=Test1234 -e SQLSERVER_DATABASE=test -e SQLSERVER_USER=test -e SQLSERVER_PASSWORD=Test1234 -p 1433:1433 exoplatfrom/sqlserver"
```

Création d'un projet

Dotnet new

La commande "dotnet new" va nous permettre de créer le projet. Si vous l'utilisez comme ça, elle va uniquement vous afficher une liste de templates disponibles :

Modèles	Nom court	Langue	Balises
Console Application	console	[C#], F#, VB	Common/Console
Class library	classlib	[C#], F#, VB	Common/Library
Unit Test Project	mstest	[C#], F#, VB	Test/MSTest
NUnit 3 Test Project	nunit	[C#], F#, VB	Test/NUnit
NUnit 3 Test Item	nunit-test	[C#], F#, VB	Test/NUnit
xUnit Test Project	xunit	[C#], F#, VB	Test/xUnit
Razor Page	page	[C#]	Web/ASP.NET
MVC ViewImports	viewimports	[C#]	Web/ASP.NET
MVC ViewStart	viewstart	[C#]	Web/ASP.NET
ASP.NET Core Empty	web	[C#], F#	Web/Empty
ASP.NET Core Web App (Model-View-Controller)	mvc	[C#], F#	Web/MVC
ASP.NET Core Web App	webapp	[C#]	Web/MVC/Razor Pages
ASP.NET Core with Angular	angular	[C#]	Web/MVC/SPA
ASP.NET Core with React.js	react	[C#]	Web/MVC/SPA
ASP.NET Core with React.js and Redux	reactredux	[C#]	Web/MVC/SPA
Razor Class Library	razorclasslib	[C#]	Web/Razor/Library/Razor Class Library
ASP.NET Core Web API	webapi	[C#], F#	Web/WebAPI
global.json file	globaljson		Config
NuGet Config	nugetconfig		Config
Web Config	webconfig		Config
Solution File	sln		Solution

Nous allons choisir de créer une application ASP.NET Core Web API, il nous faudra donc utiliser la commande "dotnet new webapi".

Vous pouvez dorénavant démarrer votre projet en utilisant la commande "dotnet run" ou "dotnet watch run".

Enfin vous pouvez ajouter une dépendance nécessaire au générateur de code :
"dotnet add package Microsoft.VisualStudio.Web.CodeGeneration.Design".

Créer des entités

- Créer d'abord un dossier à la racine du projet nommé « Entities ».
- Créer y par exemple les fichiers suivants : « CuisineType.cs » et « Value.cs »

```

public enum CuisineType {
    0 references
    None,
    0 references
    Italian,
    0 references
    French,
    0 references
    German
}

public class Value
{
    0 references
    public int Id { get; set; }
    0 references
    public string Name { get; set; }
    0 references
    public CuisineType Cuisine { get; set; }
}

```

Créer un DbContext

- Créer un dossier « Data » à la racine du projet
- Créer y votre fichier de contexte, par exemple : « ValuesDbContext.cs »

```

public class ValuesDbContext: DbContext
{
    0 references
    public ValuesDbContext(DbContextOptions options)
        : base(options)
    {
    }

    0 references
    public DbSet<Value> Values { get; set; }
}

```

Enregistrer le contexte

- Ajouter une connexion string dans l' « appsettings.json »

```

"ConnectionStrings": {
  "ValuesContext": "Server=localhost,1433;Database=test;User=test;Password=Test1234;"
},

```

- Enregistrer le contexte dans les services

```

services.AddDbContext<ValuesDbContext>(options =>
    options.UseSqlServer(Configuration.GetConnectionString("ValuesContext")));

```

Créer la première migration

Nous sommes maintenant prêts à créer la migration initiale de la DB.

Pour ce faire il faut simplement utiliser le CLI d'EntityFramework :

"dotnet ef migrations add initial"

Appliquer la migration à notre DB

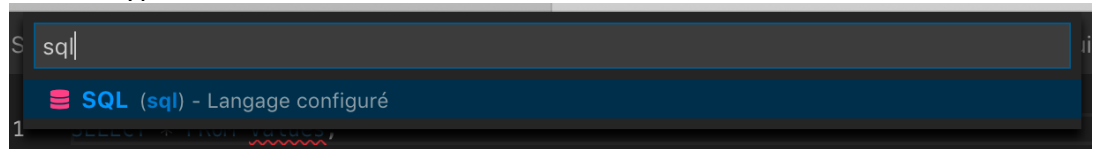
Pour appliquer la/les nouvelles migrations, vous devez utiliser la commande suivante :

"dotnet ef database update"

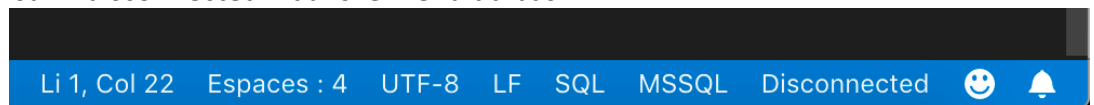
Vérifier que la DB a bien été créée

Connection

1. Créer un nouveau fichier
2. Sélectionner le type de fichier SQL



3. Cliquez sur « disconnected » dans le menu du bas



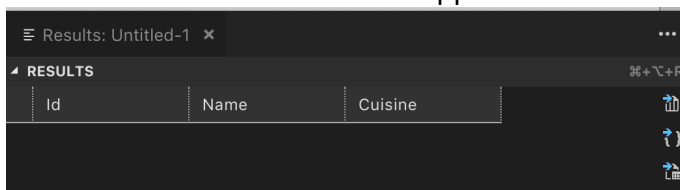
4. Entrez les informations demandées

Exécution de requêtes

1. Écrivez une requête dans ce nouveau fichier

```
SELECT * FROM [Values];
```
2. Sélectionner cette ligne
3. Appuyez sur les touches "ctrl-chift-e" ou "cmd-chift-e"

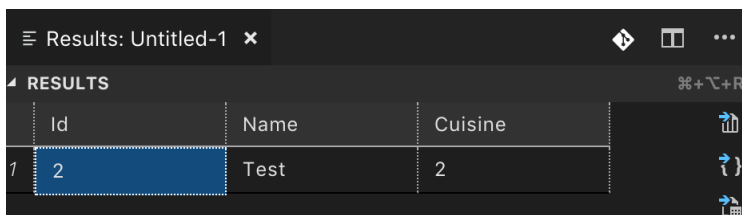
Un résultat comme celui-ci doit apparaître:



Id	Name	Cuisine
----	------	---------

Vous pouvez également insérer des valeurs par le même moyen:

```
INSERT INTO [Values] (Name, Cuisine) VALUES ('Test', 2);
```



Id	Name	Cuisine
2	Test	2

Utiliser la base de données dans un controller

Créer un service

Nous allons d'abord créer un service qui fera l'accès au DbContext.

1. Créer un dossier « Services »
2. Créer une interface, par exemple :

```

public interface IValuesData
{
    1 reference
    IQueryable<Value> GetAll();
    1 reference
    Value Get(int id);
    0 references
    Value Add(Value value);
}

```

3. Créer son implémentation

```

public class SQLValuesData : IValuesData
{
    5 references
    private ValuesDBContext _context;

    0 references
    public SQLValuesData(ValuesDBContext context) {
        _context = context;
    }

    0 references
    public Value Add(Value value)
    {
        _context.Values.Add(value);
        _context.SaveChanges();
        return value;
    }

    1 reference
    public Value Get(int id)
    {
        return _context.Values.FirstOrDefault(v => v.Id.Equals(id));
    }

    1 reference
    public IQueryable<Value> GetAll()
    {
        return _context.Values.OrderBy(v => v.Name);
    }
}

```

4. Enregistrer ce service

```

services.AddScoped<IValuesData, SQLValuesData>();

```

Utiliser le service dans le controller

1. Injecter le service

```

private IValuesData _valuesData;
0 references
public ValuesController(IValuesData valuesData) {
    _valuesData = valuesData;
}

```

2. Modifiez certaines méthodes pour utiliser le service

```
// GET api/values
[HttpGet]
0 references
public ActionResult<IEnumerable<string>> Get()
{
    return _valuesData.GetAll().Select(item => item.Name).ToList();
}

// GET api/values/5
[HttpGet("{id}")]
0 references
public ActionResult<Value> Get(int id)
{
    var result = _valuesData.Get(id);
    if (result == null) {
        return NotFound();
    }
    return result;
}
```

Créer un nouveau controller

Vous pouvez créer un controller facilement en utilisant le générateur de code :

"dotnet aspnet-codegenerator controller -name ExampleController -actions -api -outDir Controllers"

Pour d'autres utilisations que la génération d'un controller, vous pouvez aller voir sur le lien :

<https://gavilan.blog/2018/04/28/asp-net-core-2-doing-scaffolding-with-dotnet-cli-aspnet-codegenerator/>